REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 30, 1996	Final, 1 October 1992-30 May, 1996

4. TITLE AND SUBTITLE
Formal Methods for Dependable Distributed Softwarr

5. FUNDING NUMBERS G--
F49620-92-J-0546

AFOSR-TR-96
0335

6. AUTHOR(S)
B. M. McMillin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The University of Missouri-Rolla
Rolla, MO 65401

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Maj. David Luginbuhl
AFOSR/NM
110 Duncan Avenue, Suite B-115
Bolling AFB, DC 20332-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

19960627 051

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This work has developed a powerful concept in evaluating formal specifications concurrently with distributed program execution for the purposes of error detection, fault tolerance, and security. This concept is realized in the CCSP evaluation system for axiomatic proofs, for interval temporal formulae, and for a security calculus. We have validated this concept through nontrivial examples of distributed programs including a dynamic group membership protocol, a distributed database scheduler, of a responsive system modeling railroad trains on intersecting tracks, and of a secure warehouse management system. Temporal subsumption has been developed as a way of making these programs efficient. Moreover, the spinoff technologies from this work, in of themselves have become useful. CCSP can also be used as a debugging tool for distributed programs. Temporal Subsumption functions as a quick and powerful proof checker for Hoare triples. Both of these achievements may help to bring more use of formal methods into the mainstream.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Temporal Logic, Subsumption, Responsive Systems, Security, Distributed Systems	13
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	UL

Final Report
Formal Methods for Dependable Distributed Software

F49620-92-J-0546

Bruce McMillin
(ff@cs.umr.edu, Phone: (573)-341-6435, FAX: (573)-341-4501)
Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65409

Contents

1 Overview

Motivation Our original goal in this work was to find ways to execute program specifications along with the actual program's execution for purposes of run-time assurance - namely for error detection within the scope of fault tolerance. If the execution of the program does not satisfy the specification at run time, then an error has occurred. Since error detection is conceptually the most difficult problem in fault tolerance, this quantification of error detection has proved quite powerful - a

system need not rely on hardware or software confidence to avoid or detect errors; the specification provides the absolute truth of correctness.

Actually doing this is difficult even in the sequential environment as one must ask the question "What is an appropriate level of specification and how does it correspond with the resulting program code?" In the distributed parallel environment with which we are concerned, the challenge becomes greater due to the absence of a globally consistent state in which to evaluate the specification.

Methodology The notion of "the program satisfies the specification' is a powerful abstraction as it immediately draws the researcher into the area of formal logic to express the specification. This, coupled with an existing set of axioms and inference rules for a particular (programming) language provides the appropriate level of representation for run-time error checking. Essentially, the same tools used in program verification are immediately applicable to run-time assurance, namely execution of the proof outline in either a predicate or temporal framework.

Our work provides the run-time semantics to carry out such executions, possibly in the presence of failed hardware and/or software or security intrusions. Nor are we limited to formalized verification systems; our methods work quite well with informally specified assertions. We have developed a set of tools (described below) to carry out these evaluations.

2 Technical Details

2.1 The Axiomatic Approach to Program Verification

The axiomatic approach to program verification is based on making assertions about program variables before, during and after program execution. These assertions characterize properties of program variables and relationships between them at various stages of program execution.

Overall Proof Approach. Distributed programs are composed of a set of communicating sequential processes. In many programs, it is desirable to save part of the communication sequence between processes. This is done with use of "dummy" or *auxiliary* variables that relate program variables of one process to program variables of another. In general, to prove properties about the program, first properties of each component process are derived in isolation. These properties are combined to obtain the properties of the whole program using "global" auxiliary variables; if the proofs do not *interfere*, then this composition is valid. We use Hoare's CSP as a model.

Operational Evaluation of Axiomatic Assertions Taking an application's proof outline from the verification environment to the distributed operational environment

is not a straightforward task. Since assertions may involve global annotations to the program state, we need some way of communicating this state, efficiently. Observing that no state change can influence a CSP process until some communication occurs (since process states are local), we can simply defer update of global state information until an algorithmic communication occurs. Thus, each communication in CSP is augmented with two functions which prepare copies of a processes' global auxiliary variables for communication and unions these variables into a process' local state, respectively. Since we only need to send the most recent copy (and only a newer copy) variables are time stamped with a Lamport clock. Then, the latest copy of each global auxiliary variable is merged with the local processes' state. These communicated auxiliary variables, in turn, along with the sequential processes' state, are what the assertions are evaluated against.

2.2 Temporal Proof System

General Liveness Properties Using the ISTL* proof system, we constructed operational evaluation semantics of temporal specifications in the distributed environment. This enables evaluation of eventuality assertions [13]. A temporal assertion ϕ is satisfied iff for every state sequence σ of a program, σ satisfies ϕ.

Interval Assertions To achieve responsiveness, we created a new logic, ITL [12], similar to other interval logics which enable reasoning within bounded intervals, but also is amenable to embedding operational evaluation semantics of to create a temporal run-time assurance environment for liveness properties. In particular, we consider *liveness assertions* of the form $(\phi \rightarrow \mathrm{EF}\psi)$ which asserts that execution will progress from a state satisfying assertion ϕ to a state satisfying assertion ψ where ϕ and ψ are interval assertions.

Operational Evaluation As in the axiomatic system, above, the state must be communicated and time-stamped in some fashion. Since we reason about certain events in a temporal system, it is enough to collect and merge event sequences or histories during run time.

This requires two steps. First, every processor collects and orders events occurring within itself and within other processors, to form its event history. This history contains a processor's local events and its externally observable events. When a communication occurs, these histories are merged into *equivalent histories* based on *causality.* Note that this relies on neither monitors nor global clocks to compute event histories. Moreover, it does not place much additional computational or message-passing burden on the operation of the system.

Second, every processor examines its event history against assertions. Since an event history is a collection of events occurring in a system, it represents a processor's observation of all the processors during execution. This history can be utilized to do

evaluation of assertions at run-time. It is a simple matter, then, to break down the temporal assertions into predicate calculus expressions on these collected histories.

We developed the temporal logic based translation system described above and tested its ability to express specifications for the train set responsive system mentioned under CCSP [12, 14].

2.3 CCSP System

We have built the above concepts into a translation system called CCSP (C-CSP) which runs on Unix workstations which operationally evaluates axiomatic and temporal assertions embedded within a CSP-like language [4]. This code has been used within both the classroom and the research group. A user's guide exists, as well [2].

CCSP for the axiomatic proof system has been used in a course at the University of Missouri in distributed computing (C.Sc. 485). The results of this have been reported in [1]. CCSP has also been used to validate the proofs of nontrivial examples of distributed programs including a dynamic group membership protocol [10] and a distributed database scheduler [5], and a distributed sort [3]. The temporal version exists as a test version and has been used to validate the proofs of a responsive system modeling railroad trains on intersecting tracks [12].

The full CCSP source is available from `http://www.cs.umr.edu/ecl.html`

2.4 Assessment

One question that arises in building a fault-tolerant system is "How fault-tolerant is it?" In [6], we characterize degrees of fault tolerance of an error detecting algorithm based on the graphical interconnection topology of the communicating processes in a distributed system.

2.5 Assertion Generation

One of the complaints about our approach is that it requires a formal understanding of the program. In [7], we built a system to "fill in" missing assertions in a program's proof outline to be used in run-time checking. Taking this one step further, we explored the possibility of reverse engineering a program to determine its inherent safety and liveness properties by extracting the predicate transformations made by each program statement on the data domain as a symbolic expression. We then used Linear Algebraic techniques to determine invariants of the program and then took the invariants and performed an eigenvalue analysis to determine liveness constraints. Rather than generating assertions, though, we represented the information visually and built a tool to visualize program behavior [11].

2.6 Temporal Subsumption

To reduce the complexity of evaluating every assertion that arises from a proof outline, we noticed that, in fault tolerance, only those assertions which are postconditions of communication statements or make statements about auxiliary variables which might have been changed in a communication (indirectly) need to be checked. All other assertions are implied by correct execution of the local, sequential, program code. Since a processor cannot reliably reason about its own faulty or fault-free behavior, all of these other assertions are implied by their preconditions and are redundant and can be subsumed.

Temporal Subsumption, a new technique arising out of our research, is an assertion-based technique. The temporal nature comes from relating logical implications across sequences of predicate transformations described by an axiomatic proof system. Logical assertions that occur early in a program imply assertions later in the program only through predicate transformations which occur in a particular sequence, or order. Thus, unlike classical subsumption, temporal subsumption is defined with respect to the model of predicate transformations.

We have built a temporal subsumption tool using a flow graph generator, the CLP(R) proof checker, and "C programs for analysis [7].

2.7 Security

It is our supposition that formal security policies can also be executed as safety and liveness properties are. This provides an integrated methodology for ensuring that formally-specified properties of safety, liveness, fault tolerance, and security hold, at run time, in a distributed computing system in the presence of faulty hardware/software components and/or active intrusion. The underlying thread is that all violations of specifications are really errors and can be treated using an integrated methodology [8].

Our work provides the run-time semantics to carry out such execution of specifications, possibly in the presence of failed hardware and/or software and/or intrusions. Thus, the approach taken here adopts a formalized specification language together with a mechanized support tool to allow detection of certain types of errors and security breaches as described in [9].

In this application of the concept, we solved several problems.

To control the flow of information, for a particular security policy, requires that semantics and refinement be developed to generating portions of CCSP data communication layers that preserve privacy. This makes CCSP application-dependent for security uses. Additionally, in studying security we had to clip the size of the history information and developed and implemented techniques for doing so.

We also determined the algebraic requirements for a security policy language, which is amenable to encoding as first order expressions and is expressive enough to capture useful security policies.

The system was implemented as part of CCSP and tested on a model problem using an existing security calculus.

2.8 Transitions

The railroad track model problem and temporal run-time evaluation system work has yielded a contract with Harmon Electronics of Grain Valley, MO to build a railroad switching yard layout GUI tool using formal logic equations to express fault-tolerant switching routes.

The understanding of asynchronous systems has yielded joint work with Software Systems Specialists, Inc. of St. Louis, MO, and the ARMY under an STTR in performing real-time animation of manufacturing processes.

3 Awards

1994 Phillips Petroleum Foundation Faculty Excellence Award, University of Missouri-Rolla ($2500)

1993 IEEE Computer Society Certificate of Appreciation for contributions to the 1993 ICDCS

1992 Phillips Petroleum Foundation Faculty Excellence Award, University of Missouri-Rolla ($2500)

4 Invited Talks

1994 "Formal Derivation of High Assurance Concurrent Software," given at *AFOSR Software Systems Meeting, Argonne National Labs, IBM T. J. Watson Research Labs* and *The University of Idaho Department of Computer Science.*

1994 "Parallel Computing for Engineering Problems," UMR Department of Civil Engineering.

1994 "Computing: Science, Pseudo-Science, or Belief?" St. Joseph's College, Rensselaer, IN.

1993 "Parallel Algorithm Fundamentals and Analysis," *International Summer Institute on Parallel Computer Architectures, Languages, and Algorithms,* July 5-10, 1993, Prague, Czech Republic.

1993 "Computing: Science, Pseudo-Science, or Belief?" UMR Last Lecture Series.

1992 "Error-Detecting Concurrent Software through Changeling" University of Illinois-Chicago.

1992 "Assured Concurrent Software Through Application-Oriented Fault Tolerance," McDonnell Douglas Corporation, St. Louis, MO.

Personnel In addition to the following students, our group worked closely with Matt Insall of the UMR Department of Mathematics and Statistics and his students, Khanh Ngo and Anita Grogan.

5 Students Supervised

The following student theses were completed during the time period of the award. Complete copies may be obtained by sending e-mail to `csdept@cs.umr.edu` enclosing a return mailing address in the text of the message and the title and name of student of the requested document.

Student Theses - Completed				
Name	Support	Title	Degree	Grad. D
Pei-Yu Li		Fault-Tolerant Distributed Deadlock Detection	Ph.D.	1994
Jun-Lin Liu		Recoverable Ring Embeddings in Hypercubes	Ph.D.	1993
Martina Schollmeyer	AFOSR/UMR	Formal Methods for Subsumption of Assertions for Fault Tolerance in Changeling	Ph.D.	1994
Su-Mei Tsai	AFOSR	Fault-Tolerant Distributed Real-Time Systems	Ph.D.	1994
Alan Su	NSF	A Deterministic Membership Algorithm in Asynchronous Distributed Systems	M.S.	1994
Student Theses - Active				
Name	Support	Area	Degree	
Fred Budd	UMR	Distributed Computer Security	M.S.	
Jui-Lin Lu	UMR	Computational Mathematics (co-supervised)	Ph.D.	
Larry Reeves	UMR	Parallel Implicit Methods	Ph.D.	
Cristina Serban	AFOSR/UMR	Distributed Computer Security	Ph.D.	
Aggie Sun	NSF/AFOSR	Declarative Approach to Generalizing the Understanding of Prgram Behavior Through Program Visualization	Ph.D.	

6 Contribution

We feel we have developed a powerful concept in evaluating formal specifications concurrently with distributed program execution. Moreover, the spinoff technologies from this work, in of themselves have become useful. CCSP can also be used as a debugging tool for distributed programs. Temporal Subsumption functions as a quick and powerful proof checker for Hoare triples. Both of these achievements may help to bring more use of formal methods into the mainstream.

References

[1] B. Arrowsmith and B. McMillin. Teaching the practice of formal methods in distributed computing systems - a module. In *Teaching Formal Methods: Curriculum Development Workshop*, July - Aug 1994. Dept. of Computer Science, University of Missouri-Rolla, Rolla, Missouri, Technical Report CSC-94-19, July 1994.

[2] E. Arrowsmith and B.M. McMillin. How to program in CCSP. Technical Report CSC-94-20, University of Missouri - Rolla, August 1994.

[3] H. Lutfiyya, B. McMillin, and M. Schollmeyer. Fault-tolerant disributed sort generated frm a verification proof outline. In *Responsive Computer Systems - Dependable Computing and Fault-Tolerance*, pages 71–96, Tokyo, JAPAN, September 1992.

[4] B.M. McMillin and E. Arrowsmith. CCSP - a formal system for distributed program debugging. In *Proceedings of the Software for Multiprocessors and Supercomputers, Theory, Practice, Experience*, pages 260–269, Moscow, Russia, September 1994. (also as UMR Department of Computer Science Technical Report, CSC-94-13.

[5] B.M. McMillin, H. Lutfiyya, and A. Su. Formal derivation of an error-detecting distributed data scheduler using changeling. In *Lecture Notes in Computer Science Series 735*, pages 363–376. Springer-Verlag, July 1993. (also as a poster in the 15th ICSE, Baltimore, MD, May, 1993).

[6] M. Schollmeyer and B.M. McMillin. A general method for maximizing the error-detecting ability of distributed algorithms. In *PARLE '94 Parallel Architectures and Languages Europe*, pages 725–736, Athens, Greece, July 1994. Springer-Verlag, Berlin.

[7] M. Schollmeyer and B.M. McMillin. Checking program proofs made easy. In *Proceedings of COMPSAC'95, the 19th International Computer Software and Applications Conference*, pages 102–107, August 1995.

[8] C. Serban and B.M. McMillin. From formal security specifications to executable assertions - a distributed systems preliminary study. Technical Report CSC-95-01, University of Missouri - Rolla, April 1995.

[9] C. Serban and B.M. McMillin. Run-time security evaluation (RTSE) for distributed applications. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 222–232, Oakland, CA, May 1996.

[10] Alan Su. *A Deterministic Membership Algorithm in Asynchronous Distributed Systems.* PhD thesis, University of Missouri - Rolla, May 1994.

[11] A. Sun, , and B.M. McMillin. Wheels: An automated program analysis tool. In *Proc. 8th International Conference of Software Engineering and Knowledge Engineering*, Lake Tahoe, NV, June 1996. Knowledge Systems Institute.

[12] G. Tsai, M. Insall, and B. McMillin. Constructing an interval temporal logic for real-time systems. In *Proc. First IEEE International Conference on Engineering of Complex Computer Systems*, November 1995. also as UMR Department of Computer Science Technical Report Number CSC 93-25).

[13] G. Tsai, M. Insall, and B.M. McMillin. Ensuring the satisfaction of a temporal specification at run-time. In *Proceedings of the First IEEE International Conference on Engineering of Complex Computer Systems*, pages 397–404, Ft. Lauderdale, FL, November 1995. (also as UMR Department of Computer Science Technical Report Number CSC 93-20).

[14] S.G. Tsai. *Providing Assurance for Responsive Computing Systems.* PhD thesis, University of Missouri - Rolla, May 1994.

7 Other Publications Resulting From the Contract

15. Li, P.-Y. and McMillin, B., "Formal Model and Specification of Deadlock," CSC-93-31, University of Missouri - Rolla, August, 1993

16. Li, P.-Y. and McMillin, B., "Formal Verification of Distributed Deadlock Detection Algorithm Using a Time-dependent Proof Technique," CSC-94-06", University of Missouri - Rolla, Feb, 1994

17. "An Improved Characterization of 1-step Recoverable Embeddings: Rings in Hypercubes," Proceedings of the 1994 International Conference on Parallel and Distributed Systems, (with J. Liu and T. Sager), pp. 508-513, December 1994.

18. Smolik, T., McMillin, B., Krishnamurthy, K. and Lu, W., "An Approach for Implementing On-Line Learning for Force Control in End Milling Operations," *Dept. of MAE&EM, University of Missouri-Rolla, Rolla, Missouri, Technical Memorandum MAE-TM-30*, August 1994.

19. Su, A., Swope, K., Krishnamurthy, K., Lu, W. and McMillin, B., "Experimental Setup to Test Intelligent Control Algorithms for Controlling Cutting Force in End Milling Operations," *Dept. of MAE&EM, University of Missouri-Rolla, Rolla, Missouri, Technical Memorandum MAE-TM-29*, May 1994.

20. Xu, Q., Krishnamurthy, K., McMillin, B. and Lu, W., "A Recursive Least Squares Training Algorithm for Recurrent Neural Networks," *Proceedings of the American Control Conference* , pp. 1712-1716, June 1994.

21. Xu, Q., Krishnamurthy, K., McMillin, B. and Lu, W., "Identification of Cutting Force in End Milling Operations Using Recurrent Neural Networks," *Proceedings of the IEEE Conference on Neural Networks* , pp. 3828-3833, June-July 1994.

22. Luo, T., Xu, Q., Krishnamurthy, K., Lu, W. and McMillin, B., "Force Control in Two-Dimensional End Milling Operations Using Recurrent Neural Networks," *1995 ASME Int. Mechanical Engineering Congress and Exposition*, DSC-Vol. 57-2, November 1995, pp. 773-780.

23. "CCSP - A Formal System for Distributed Program Debugging," Programming and Computer Software, Plenum Publishers, (to appear) (B. McMillin and E. Arrowsmith).

24. "Relaxing Synchronization in Distributed Simulated Annealing," IEEE Transactions on Parallel and Distributed Systems (B. McMillin and C. Hong), Vol. 6, No. 2, pp. 189-195.

25. "A Neural Controller for Force Control in End Milling," 1994 International Mechanical Engineering Congress and Exposition, (B. McMillin, K. Krishnamurthy, Q. Xu, and W. Lu), DSC-Vol. 55-1, pp. 563-572, November, 1994,

26. "A Recursive Least Squares Training Algorithm for Multilayer Recurrent Neural Networks," *Proc. of the American Control Conference,* (B. McMillin, K. Krishnamurthy, Q. Xu, and W. Lu), pp. 1712-1716, June, 1994.

27. "An Enhanced Reconfigurable Embedding Scheme for Rings in Hypercubes," *Proceedings of the 1992 International Conference on Parallel and Distributed Systems,* (B. McMillin, J. Liu), pp. 298-305, December, 1992.

28. "The Computation of Supersonic Combustor Flows using Multi-Computers," *American Institute of Aeronautics and Astronautics (AIAA)*, AIAA paper number 93-0060, Reno, NV, January, 1993, (B. McMillin, D. Riggins, M. Underwood, L. Reeves, and E. Lu).

29. "Modeling of Supersonic Combustor Flows Using Parallel Computing," *Computer Systems in Engineering,* Pergamon Press Ltd., Great Britain, Vol. 3, Nos 1-4, pp. 217-229, 1992, (B. McMillin, D. Riggins, M. Underwood, L. Reeves, and E. Lu).

30. "A Divide and Conquer Ring Embedding Scheme on Hypercubes with Efficient Recovery Capability," *Proceedings of the 21st International Conference on Parallel Processing,* pp. III-38-III-45, August, 1992 (B. McMillin and J. Liu).

31. "Fault-Tolerant Distributed Deadlock Detection/Resolution," *Proceedings of the 17th International COMPSAC,* November, 1993, pp. 224-230, (B. McMillin and P. Li). (Also as *UMR Department of Computer Science Technical Report Number CSC 92-04*).

32. "Fault-Tolerant Concurrent Branch and Bound Algorithm Derived from Program Verification" *Proceedings of the 16th International COMPSAC,* September, 1992, pp. 182-187, (B. McMillin, A. Sun and H. Lutfiyya).

33. "Parallel Algorithm Fundamentals and Analysis," *Parallel Computer Architectures, Languages, and Algorithms,* ed. T. Casavant,, P. Tvrdik, and F. Plasil, IEEE Computer Society Press, 1995, pp. 151-182. (B. McMillin and J. Liu).

34. "Formal Methods to Generate and Understand Distributed Computing Systems," *Parallel Computer Architectures, Languages, and Algorithms,* ed. T. Casavant, P. Tvrdik, and F. Plasil, IEEE Computer Society Press, 1995, pp . 351-368, (B. McMillin, G. Tsai and H. Lutfiyya).

35. "Modeling of Injection in Supersonic Combustor Flows," *1994 SCALABLE HIGH PERFORMANCE COMPUTING CONFERENCE,* Knoxville, TN, May 1994. Poster Presentation (B. McMillin, Mark Underwood, David Riggins, Larry Reeves, and Eric Lu)

36. "Formal model and specification of deadlock," *UMR Department of Computer Science Technical Report CSC-93-31* (B. McMillin and P. Li) .

37. "Formal verification of distributed deadlock detection algorithm using a tim e-dependent proof technique," *UMR Department of Computer Science Technical Report Number CSC-94-06* (B. McMillin and P. Li).

38. Schollmeyer, M. and McMillin, B., "Using Temporal Subsumption for Developing Efficient Error-Detecting Distributed Algorithms," *Dept. of Computer Science, University of Missouri-Rolla, Rolla, Missouri, Technical Report CSC-93-28,* October 1993.

39. Schollmeyer, M. and McMillin, B., "Efficient Run-Time Assurance in Distributed Systems Through Selection of Executable Assertions," *Dept. of Computer Science, University of Missouri-Rolla, Rolla, Missouri, Technical Report CSC-94-1*, January 1994.

Copies of the above technical reports may be obtained from `http://www.cs.umr.edu/techreport`